APPLE

COPILOT

B KEEPES

COPILOT

AN AUTHOR LANGUAGE FOR WRITING CAI LESSONS

APPLE II VERSION - 2.2.1

BRUCE KEEPES
DEAN ROSENHAIN
JEFF WIGNALL

1982

Cover design by
Peter McWilliams

South Australian College of Advanced Education
Magill, South Australia, 5072

Third Edition - August 1982

COPILOT is made available at a low cost for non-profit uses as a service to the educational community. As such, it is not intended as a commercial product and is sold on an 'as is' basis. No warranty, either expressed or implied, is made with respect to the performance or fitness for any particular purpose of the computer programs and written materials.

The authors are, however, desirous of both correcting any faults and continuing to improve the programs and documentation. Any comments or suggestions should be directed to Dr. Bruce Keepes, South Australian College of Advanced Education, Magill, South Australia, 5072. Likewise, information on purchase of COPILOT may be directed to Dr. Keepes.

- - - - - - -

TABLE OF CONTENTS

INTRODUCTION

What is COPILOT?

COPILOT is an author language designed for use by teachers writing computer assisted instruction (CAI) lessons, one of the many ways in which the computer can be used in the learning situation. As an author language, it is elegant, and powerful. It consists of only eleven basic commands, but possesses a large number of powerful features. Among these are: selective branching to route students through different materials according to their responses, storage of student responses for later insertion in the lesson, search for a selected word in a student response, and graphics and sound.

And it works! Teachers with no previous programming experience have learned to write their own CAI lessons in reasonably short periods of time.

Why a special language for CAI?

COPILOT, like other author languages, grew out of the conviction that if a teacher wrote a CAI lesson, and then had a computer programmer write the computer program, his/her biases would creep into the lesson, distorting to some extent the teacher's intentions. Further, once a lesson was programmed, it would tend to remain fixed since the teacher would need a programmer's services in order to make modifications.

What was desired then, was an author language which all teachers could learn in a short period of time and could use to write a wide variety of lessons for their students.

Building on the experiences of others, COPILOT was developed as a simple, powerful language. Of course it has its omissions - for example, not keeping a statistics file on students, and not having a random number generator as a part of COPILOT. Both of these omissions were diliberate: we felt that emphasis should be on the learning aspects of a lesson rather than grading a student; and we believe that the sequence of problems should be specified precisely rather than randomly generated.

How does COPILOT work?

The teacher will come in contact with two aspects of the COPILOT language - the COPILOT program itself, which runs the lesson, and the COPILOT Editor (COPED) and Shape Maker by which the lessons are written, edited, and revised.

The student will only contact the COPILOT program, since he makes no change to the content of the lesson - that editing function is reserved for the teacher. Any number of students may run the lesson at the same time using the COPILOT mode. To each student, it seems that the computer is communicating directly with him, responding to his answers and requests for hints.

Actually, the student is simply progressing through a lesson in the computer, designed by the teacher to provide certain responses to certain answers to the teacher's questions.

The following sections in this manual explain the fundamental COPILOT commands which are used in constructing a lesson, the editing commands with which to manipulate them, and the procedure of running a lesson.

How did COPILOT originate?

In 1971 the Palo Alto Unified School District (California) received a research grant from the United States Office of Education for the development of a series of computer assisted instruction lessons for use with acoustically handicapped primary and secondary students. One of the decisions made by those directing the project was that the special education teachers who taught the students would write the computer programs for the lessons. However, none of these teachers had any skills in computer programming. This, plus the difficulty of writing CAI lessons directly in FORTRAN (the language of the School District computer), necessitated either the adoption of an existing, or the development of a new, "author" language specifically designed for writing CAI lessons.

The language PILOT, which had been developed at the University of California, was in its infancy and was viewed as a likely adoption. However, it was not fully developed and would need to be modified for the Palo Alto project. Further, the differences between the computer on which PILOT was developed and the School District computer meant a major rewriting of the computer code. It was, therefore, decided to write a new author language based on PILOT. To acknowledge its origins, it was decided to call this new language COPILOT.

Since these early beginnings COPILOT has had a number of features added and has been modified to work on a number of different computers - far too many to mention. Two programs, COPILOT and COPED, have had so many patches that they are now prime examples of completely unstructured programming. Neither Aram Grayson, who wrote the original code, nor Jeff Wignall, who did the modification for the Apple II, should in any way be held responsible for this situation. The SHAPE MAKER is a new feature which was developed by Dean Rosenhain; this feature was not implemented on previous versions of COPILOT.

PART I

THE COPILOT AUTHOR LANGUAGE

This section describes the fundamental COPILOT commands that are used to put a lesson together from text and questions, and to handle student input within the lesson.

What is a lesson going to look like?

First of all, very different to teacher and student. The teacher will write a lesson allowing for various types of student answers. The lesson the student sees will be written 'on the spot' according to his answers. For example, a right response will send him on one path, a wrong response will send him on another path through that particular section of material.

Here is a part of a simple lesson on the use of "doesn't" and "don't" as written by the teacher.

```
 90 Q.A MONKEY ----- HAVE A SHORT TAIL.
100 L.1
110 A.DOESN'T
120 R.VERY GOOD
130 W.TRY AGAIN
140 WX1
150 W.THE ANSWER IS:  DOESN'T
160 Q.
170 Q.ZEBRAS ----- HAVE SPOTS.
180 L.2
190 A.DON'T
200 R.VERY GOOD
210 W.TRY AGAIN
220 WX2
230 W.THE ANSWER IS:  DON'T
240 Q.
```

Here are two runs. One by a student who gave a right answer, one by a student who gave a wrong answer:

```
A MONKEY ----- HAVE A SHORT TAIL.
?DOESN'T
VERY GOOD

ZEBRAS ----- HAVE SPOTS.
?DON'T
VERY GOOD
```

```
A MONKEY ----- HAVE A SHORT TAIL.
?DON'T
TRY AGAIN
?DOESN'T
VERY GOOD

ZEBRAS ----- HAVE SPOTS.
?DOESN'T
TRY AGAIN
?DON'T
VERY GOOD
```

As you can see, the student is apparently 'talking with' the computer in everyday English. The teacher's lesson is more complicated, since it must provide the many different versions appropriate to different student responses. Each line of the teacher constructed lesson consists of two parts: a line number and a two character *command* (e.g. Q.); and, in most cases, a line of text. It is the two character command that tells the computer what to do with the rest of the line or which line to execute next, and it is in this structure that the lesson is written.

So the structure of the lesson is actually as follows:

```
 90 Q.
100 L.1
110 A.
120 R.
130 W.
140 WX1
150 W.
160 Q.
170 Q.
180 L.2
190 A.
200 R.
210 W.
220 WX2
230 W.
240 Q.
```

What are the functions of the line numbers and commands?

The line numbers (10, 20, 30, etc.) are supplied automatically by the computer and provide the teacher with a convenient way to edit text. For example, if the teacher wishes to delete line 40 from the lesson, she could simply type D40 (D is for delete). If she wished to insert a line between lines 10 and 20, she could type E15 (E is for enter, and 15 is between 10 and 20).

The series of two character *commands* provides instructions to the computer.

For example, Q. tells the computer to print the text following the Q. exactly as it appears. The A. tells the computer to do two things: stop and wait for the student to type in a response; and compare that response with the 'right' answer which the teacher has entered following the A. to determine whether the student's response is correct.

Here we need to make a distinction. The COPILOT commands (Q. A. W. R. etc.) are instructions to the computer when a student is running a lesson; each line of the lesson begins with one of these two character commands. The COPED commands (E D L U R and T) are used by the instructor in writing the lesson; they allow the instructor to Enter lines, Delete lines, List the lesson, etc.

This section explains the basic COPILOT commands, and how a teacher can use them to construct a variety of lessons. The next section, The COPILOT Editor, explains how the teacher inputs the material into the computer.

Q.

(the quote or question command)

Whatever is typed after the Q. will be printed when the student runs the lesson. This might be an introductory line or paragraph of instructions, or a question.

If more than one line is needed, each line must be preceeded by a Q.

Example:

```
10 Q. CHRISTOPHER COLUMBUS SAILED FROM ITALY IN 1492 AS
20 Q. CAPTAIN OF THE NINA; THE PINTA; AND THE SANTA MARIA;
30 Q. AND LANDED ON THE SHORES OF A CARRIBEAN ISLAND
```

This will be printed for the student as:

```
CHRISTOPHER COLUMBUS SAILED FROM ITALY IN 1492 AS
CAPTAIN OF THE NINA; THE PINTA; AND THE SANTA MARIA;
AND LANDED ON THE SHORES OF A CARRIBEAN ISLAND
```

COMMENT

Q. may be used to achieve line spacing by typing Q. and then a carriage return.

The line numbers (10, 20, 30) are supplied by the computer during editing. These are discussed in part 2. When the student runs the lesson the line numbers and commands are not printed.

In the example above, we left a space between the Qs and the text following. In the example on page 4 we did not. Both will work. The only effect of the space following the Q. (and the W. and R. for that matter) will be to leave a space at the beginning of the line when it is printed.

For those persons familiar with the computer language BASIC, <Q.> is equivalent to PRINT "....."

A. B.

(the answer commands)

The A. is a two-part command. First, it causes the computer to print a ? mark and wait for the student to respond. Second, after the student responds, it compares his input with whatever the lesson-writer has typed after the A.

If the two are the same, a condition of 'rightness' is established. If they are not the same, a condition of 'wrongness' is established. Just what will be printed out under these conditions is determined by the teacher using the R. and W. which follow on the next pages.

If the teacher chooses the default option, that is, not to specify a response, the computer will print 'RIGHT' or 'WRONG' depending on the existing condition.

Example:

```
10 Q. WHO DISCOVERED AMERICA IN 1492?
20 A. CHRISTOPHER COLUMBUS
30 A. COLUMBUS
40 A. C. COLUMBUS
```

(In this case the teacher anticipates a number of versions of the 'correct' answer, and so writes an A. line for each one. The computer will compare the student's response with all adjacent A. lines, and will print 'WRONG' only if the student's response does not match with any one of them.)

This will run for the student as follows:

```
WHO DISCOVERED AMERICA IN 1492?      WHO DISCOVERED AMERICA IN 1492?
?LEIF ERICSSON                  or   ?CHRISTOPHER COLUMBUS
WRONG                                RIGHT
```

COMMENT

The ? on the second line is supplied by the computer to signify the student is to type in a response.

The A. command ignores blanks in a student response. If these are important to recognize, the programmer should use B. command, which operates just like A. except that both characters and blanks are compared.

For those familiar with BASIC, <A. string> is equivalent to

```
INPUT A$
R1=0
IF A$ = string THEN R1 = 1
```

where R1 is a 'rightness' condition variable which is initially set to 0 (the condition of 'wrongness') and subsequently set to 1 if the student input (A$) matches the string following the A.

R.

(rightness response)

Any text following the R. will be printed if the student's response is evaluated as right.

Example:

```
10 Q. WHO DISCOVERED AMERICA?
20 A. COLUMBUS
30 R. GOOD!
```

This will run as follows:

```
WHO DISCOVERED AMERICA?
?COLUMBUS
GOOD!
```

If no R. statement is written into the lesson, COPILOT will automatically respond 'RIGHT' to a correct answer through what is called the 'default option'. This option may be suppressed by using the command R. with no text following. That is, an R. followed by a blank line simply prints a blank line.

Once an R. is executed, it will not be executed again until a Q. is executed (see p.14 for an example).

For those familiar with BASIC, <R. string> is equivalent to

IF R1 = 1 THEN PRINT 'string'

except that a separate counter will cause the statement to be skipped if (a) the R. has already been executed and (b) a Q. was not executed since the R. was executed.

W.

(wrongness response)

This is similar to the R. command, printing a specified text when the student gives a response which is evaluated as wrong.

W. may be followed by a statement like 'incorrect, try again', or may give a clue to the correct answer.

Example:

```
10 Q. WHO DISCOVERED AMERICA?
20 A. COLUMBUS
30 R. GOOD!
40 W. THAT IS NOT THE PERSON I WAS THINKING OF.  I
50 W. WAS THINKING OF AN ITALIAN WHOSE NAME STARTS WITH
60 W. A C.  TRY AGAIN.
```

A student answering 'Columbus' would receive the run on the previous page. A student answering incorrectly, would receive the run below.

```
WHO DISCOVERED AMERICA?
? LEIF ERICCSON
THAT IS NOT THE PERSON I WAS THINKING OF.  I
WAS THINKING OF AN ITALIAN WHOSE NAME STARTS WITH
A C.  TRY AGAIN.
```

The default option for W. is 'WRONG'; if there is no W. listed and the student's response is evaluated as incorrect the word 'WRONG' will be printed. To suppress the WRONG statement put a W. followed by a blank line in the lesson.

Once a W. is executed, that line will not be executed again until a Q. is executed (see page 14 for an example).

For those familiar with BASIC <W.> is equivalent to

```
IF R1 = 0 THEN PRINT 'string'
```

except that as with R. a separate counter will cause the statement to be skipped if (a) the W. has already been executed and (b) a Q. was not executed since the W. was executed.

Z.

(the snooze command)

Provides for a line of student input, but does not evaluate it.

As many contiguous lines as desired may be designated as Z. but the student should be notified in the text how many lines will be allowed for his response, since he must give exactly that number of responses.

Example:

```
10 Q. GO TO THE FRONT OF THE ROOM
20 Q. AND PULL DOWN THE MAP.
25 Q. TYPE ANY CHARACTER AND
30 Q. PRESS THE CARRIAGE RETURN KEY WHEN YOU ARE READY.
40 Z.
50 Q. NOW TAKE YOUR STUDY SHEET AND UNDERLINE THE
60 Q. NAME OF THE STATE COLOURED GREEN.
```

In this example lines 10 to 30 are typed. The machine then waits for the student to respond in line 40, and after that input continues with the lesson at line 50.

COMMENT

The student may also choose not to respond, but to type 'no comment' and carriage return or even just a carriage return. Since his response is not evaluated, the lesson will simply continue.

The Z. command can be used to allow a student to type in, for example, his or her name, have it stored using the S.n command (see apge 16), and have it printed out later in a lesson.

@.....@

KEYWORD SEARCH

Whatever the author places between @ signs will be evaluated as correct if it appears anywhere in the student's response.

Example:

```
10 Q. WHO DISCOVERED AMERICA?
20 B.@COLUMBUS@
30 R. GOOD!
40 W. NOT TOO GOOD...
```

Any student input including 'COLUMBUS' will be evaluated as correct and receive the following output:

```
WHO DISCOVERED AMERICA?
?COLUMBUS
GOOD!
```

If the teacher specified @ COLUMBUS@ 'CHRISTOPHERCOLUMBUS' would be considered wrong, since it lacks a space before the 'COLUMBUS', while 'IT SURE WASN'T COLUMBUS' would be considered right.

COMMENT

The first @ sign must be immediately after the punctuation mark following the line command.

For example: `40 A.@COLUMBUS@`

not: `40 A. @COLUMBUS@`

Through the use of the keyword search it is possible to evaluate a student's response without regard to word order. For example:

```
10 Q. MARY HIT THE BALL
20 Q. LIST ALL OF THE NOUNS IN THIS SENTENCE.
30 A.@MARY@BALL@
```

If the student typed either 'MARY BALL' or 'BALL MARY' his/her answer would be evaluated as correct.

Those familiar with BASIC will find it an interesting exercise to determine how many BASIC statements would be required to program the COPILOT statement

A.@STRING@

Alternatively, how many BASIC statements would be required for

A.@STRING1@STRING2@STRING3@?

L.

(location command)

L. establishes a location in a lesson. It is used in conjunction with the X., RX, and WX commands on the following pages.

L. is always followed by a number from 1 to 99 to establish a location in the lesson (for example, L.2).

The L. number is not the same as the line number which appears before each command. An L. line might read: 45 L.3

The 45 is only a convenience for entering lines (writing a lesson) and editing. The 3 establishes the location within the lesson.

COMMENT

Along with the X., RX, and WX commands on the following pages, this command provides for branching, or routing the student to subroutines.

X. WX RX

(the x-country or transfer commands)

X. means to jump to another location in the lesson. It is always followed by a location number from 1 to 99.

The command X.2 jumps in the lesson to the statement following L.2.

RX is used to execute a jump when a match is made and the student's resonse is evaluated as 'RIGHT'.

WX is used to execute a jump when a match is made and the student's response is evaluated as 'WRONG'.

Both must be followed by a location number from 1 to 99.

Note that the command is WX3, not WX.3.

COMMENT

Once a WX or RX has been executed it will not be executed again until a Q. or L. is executed. Consider the following two examples:

INCORRECT

```
10 Q. AN ADDITION LESSON.
15 L.1
20 Q. WHAT IS 2 + 2
30 A.4
40 R. GOOD ON YOU
50 W. WRONG, TRY AGAIN
60 WX1
70 W. STILL WRONG
80 W. THE ANSWER IS 4
90 Q. LESSON CONTINUES
```

CORRECT

```
10 Q. AN ADDITION LESSON
20 Q. WHAT IS 2 + 2
25 L.1
30 A.4
40 R. GOOD ON YOU
50 W. WRONG, TRY AGAIN
60 WX1
70 W. STILL WRONG
80 W. THE ANSWER IS 4
90 Q. LESSON CONTINUES
```

In the 'incorrect' example, if the student does not know the answer he/she will be in an endless loop with the computer always stating 'WRONG, TRY AGAIN'. In the 'correct' example, if the student answers incorrectly the first time he/she will be given a second try. If the answer is still incorrect lines 50 and 60 will be skipped (not executed) and the student will be given the answer in lines 70 and 80.

The X., WX and RX must not have anything else on the line except the label number.

For those familiar with BASIC <X.n> is equivalent to GOTO n while <RXn> and <WXn> are equivalent to IF .... GOTO or IF .... THEN.

X.0

(SUBROUTINES)

X.0 (X dot zero) will cause a transfer to the next statement after the last transfer.

Example:

```
 10 Q.                EXPLORERS
 20 Q. THIS IS A LESSON ABOUT THREE FAMOUS EXPLORERS
 30 X.1
 40 Q. WHO DISCOVERED AMERICA?
 50 A.@COLUMBUS@
 60 R. VERY GOOD
 70 W. THE CORRECT ANSWER IS COLUMBUS; SHALL WE TRY ANOTHER
 80 Q. WHICH ONE OF THESE WAS ITALIAN?
 90 X.1
100 A.@COLUMBUS@
110 R. GOOD FOR YOU
120 W. DRAKE WAS ENGLISH; MAGELLAN PORTUGUESE; ONLY COLUMBUS WAS ITALIAN
130 X.5
140 L.1
150 Q. COLUMBUS           DRAKE           MAGELLAN
160 X.0
170 L.5
180 Q. LESSON CONTINUES
```

COMMENT

In the example above, the words Columbus Drake Magellan will be first printed out after the sentence 'THIS IS A LESSON...'. Later it will be printed out after the statement 'WHICH ONE OF THESE...'. In both cases the X.0 causes transfer back to the statement after the X.1 (line 30 in the first case, line 90 in the second).

For those familiar with BASIC, X.0 is equivalent to

RETURN

where the RXn, WXn, or X.n is treated as GOSUB.

A, B,

(evaluate the last student response)

A, ignores blanks, does not wait for a reply.
B, compares both characters and blanks; does not wait for a reply.

Example of text:

```
10 Q. WHO DISCOVERED AMERICA IN 1492?
15 L.2
20 A. COLUMBUS
25 RX1
30 A, LEIF ERICCSON
40 R. LEIF ERICCSON IS NOT WHO I WAS THINKING
45 R. OF.  TRY AGAIN
50 RX2
60 L.1
70 Q. NEXT QUESTION
```

COMMENT

Sometimes it is desirable to evaluate the last response input by the student (see example above). The A, permits this to be done.

In the example given, if the student were to respond 'COLUMBUS' at line 20, the response would be evaluated 'RIGHT' and the student would be transferred to a place called L.1 in the program through the command RX1. If the student does not respond 'COLUMBUS' his response will be compared to 'LEIF ERICCSON' by the A, command at line 30.

This example illustrates one way of providing a response to an anticipated wrong answer (a 'wrong' answer which the teacher anticipates the student might make). If the student responds LEIF ERICCSON, the response 'LEIF ERICCSON WAS NOT...' is given, the program causes transfer back to label 2 (L.2), and the student has another chance to respond.

S. U. U, Q, R, W,

To Save a Student Response and Utter it Later

S.n Save the last student response in a location called n. n must be an integer from 1 to 9 (a maximum of nine different responses may be saved at any one time).

U.n Utter (type) the student response which was previously saved in location n.

U,n Works the same as U.n except there is no carriage return or linefeed after printing.

Q, Works the same as Q. except there is no carriage return or linefeed after printing.

R, Works the same as R. except there is no carriage return or linefeed after printing.

W, Works the same as W. except there is no carriage return or linefeed after printing.

Example:

```
 10 Q.HELLO
 20 Q.WHAT'S YOUR NAME?
 30 Z.
 40 S.1
 50 Q.WHAT'S YOUR TEACHER'S NAME?
 60 Z.
 70 S.2
 80 Q,WELCOME,
 90 U,1
100 Q,, TO
110 U,2
120 Q.'S COMPUTER CLASS.
```

The lesson will execute in the following fashion:

```
  HELLO
  WHAT'S YOUR NAME?
? ARAM
  WHAT'S YOUR TEACHER'S NAME?
? BRUCE
  WELCOME, ARAM, TO BRUCE'S COMPUTER CLASS.
```

EXPLANATION

The Z in line 30 allows the student to enter his name. The S.1 in line 40 stores that information in location 1 (called a buffer location). Similarly line 50 allows the student to enter his teacher's name; 70 stores it in buffer location 2. Note that the correct spacing must be inserted by the teacher in the Q, and Q. lines.

THIS NEXT EXAMPLE WILL SAVE A STUDENT'S RESPONSE IF IT'S WRONG

```
 10 Q.WHO DISCOVERED OODNADATTA?
 20 L.1
 30 A.FRED
 40 R.WHAT A BRAIN!
 50 RX2
 60 S.1
 70 W,NO,
 80 U,1
 90 W.IS WRONG
100 W.TRY AGAIN
110 WX1
120 W,IT'S NOT
130 U,1
140 W.EITHER
150 W.TRY ONCE MORE.
160 WX1
170 W,THE ANSWER IS 'FRED', NOT
180 U,1
190 W.AS YOU SAID.
200 L.2
210 Q.NEXT QUESTION
```

EXPLANATION: THERE IS NO NEED FOR A Z. BECAUSE THE RESPONSE BUFFER GETS FILLED AS A RESULT OF THE A. The following two examples illustrate the sequence which would occur for two different student responses.

Example 1:

```
 WHO DISCOVERED OODNADATTA?
?FRED
 WHAT A BRAIN!
 NEXT QUESTION
```

Example 2:

```
 WHO DISCOVERED OODNADATTA?
?SAM
 NO, SAM IS WRONG
 TRY AGAIN
?GEORGE
 NO, IT'S NOT GEORGE EITHER
 TRY AGAIN
?MABEL
 THE ANSWER IS 'FRED', NOT MABEL AS YOU SAID.
 NEXT QUESTION
```

C.

C. chains to another lesson.

Example:

```
670 Q.THAT COMPLETES THIS LESSON.  WOULD YOU LIKE TO TRY ANOTHER?
680 A.YES
690 W.
700 WX99
710 R.OK, HERE WE GO!!!!....
720 C.TAXES
730 L.99
740 Q.SEE YOU LATER, PINK PERTATER........
```

COMMENT

This is an example of a transfer at the end of a lesson. If the student responds 'Yes' in response to the ? generated by line 680 the response in line 710 is printed and line 720 chains to the lesson titled TAXES. The student will then begin taking that lesson. If the student responds anything else to the ? line 700 will cause a transfer around the C. command and the lesson ends.

It is also possible to enter the middle of a lesson by typing a semi-colon and a location number after the lesson name. For example:

```
540 C.TAXES;3
```

would chain to location 3 in the program called TAXES.

NOTE: THE MAXIMUM NUMBER OF LINES A LESSON MAY CONTAIN IS 500. IF YOUR LESSON IS LONGER THAN THAT, BREAK THE LESSON INTO SECTIONS AND USE THE CHAIN COMMAND.

SUMMARY - FUNDAMENTAL COPILOT COMMANDS

Q. Causes the line of text to be displayed.

Q, Same as Q. except there will be no line-feed for next line.

A. Pause, compare against student's response, ignore blanks and case.

A, Like A. but does not pause.

B. Like the equivalent A but compares blanks.

B, Like the equivalent A but compares blanks.

L.n Establish a numbered label in the lesson.

R. Type message if answer is 'Right'.

RXn Conditional jump to n if 'Right'.

S.n Stores the students last response in buffer n.

U.n Prints the contents of buffer n.

W. Type message if answer is 'Wrong'

Wxn Conditional jump to n if 'Wrong'

X.n Unconditional jump to n.

Z. Accept student response but do not evaluate it.

C. Chains to another lesson.

PART II

THE COPILOT EDITOR COMMANDS

The following pages explain the mechanics of entering, correcting, and checking statements using the COPilot EDitor mode.

TO WRITE A NEW LESSON

COMMENT

This procedure may look complicated, but after you have done it a few times it becomes quite easy and logical. For clarity on the next two pages the computer prompts are in capitals, not underlined, while what you respond is in capitals, underlined.

1. Get access to the COPilot EDitor, in which you will write the lesson, by typing:

   RUN COPED

2. The computer will type WHAT IS THE NAME OF YOUR LESSON?

   Type in the name by which you want your lesson to be known. It can be any combination of up to twelve characters.

3. The computer will type IS IT OLD OR NEW?

   Type NEW if the lesson is a new one you are writing or OLD if it already exists and you are editing it. N.B. If you type NEW, any lesson or program on the disk with that name will be destroyed.

4. If you specify that your lesson is OLD, the computer will type:

   WHAT NAME DO YOU WISH TO GIVE TO THE MODIFIED LESSON?

   At this point you can give a new name to your lesson, which preserves the original copy as a backup.
   Computer responds

   LESSON <LESSON NAME> IS BEING LOADED.

   At this point there will be a 10-45 second delay while the lesson is loaded from disk.

5. The computer will type OPERATION:?

   You can then begin writing your lesson by typing

   E

   (E and other edit commands are discussed on the following pages. We thought it best to provide a general overview first and then go into specifics.)

   and the computer will respond 10

   asking you for a line command.

6. When you finish writing the lesson, type / in response to the line number prompt. This will remove you from the enter phase. (N.B. There will be a 5-15 second delay at this point - do not panic!)

   The computer will respond OPERATION:?

   Type / again and the computer will take you out of the edit mode. Again there will be a 5-15 second delay. For example:

   OPERATION:? /

7. If the computer now types <NAME> IS YOUR LESSON NAME
   the lesson is saved in the computer.

8. The computer will then type DO YOU WANT TO RUN THE EDITOR AGAIN?
   Type YES or NO
   If you type 'YES' the program will return to step 2 and allow you to edit another lesson.

9. If you type 'NO' the computer will type
   DO YOU WANT TO RUN YOUR LESSON?
   Type YES or NO
   If you type 'NO' the computer will type
   GOODBYE AND GOOD LUCK WITH YOUR LESSON.

10. If you type 'YES' the computer will type
    OK, WAIT A MINUTE ....
    and transfer you to the program COPRUN to run your lesson.

TO EDIT AN EXISTING LESSON

1. Get access to COPED, the COPilot EDitor,

by typing	RUN COPED
The computer types	WHAT IS THE NAME OF YOUR LESSON?
You type	<NAME OF LESSON>
The computer types	IS IT OLD OR NEW?
You type	OLD
Computer types	WHAT NAME DO YOU WISH TO GIVE TO YOUR MODIFIED LESSON?
You type	<NAME OF LESSON>
The computer types	OPERATION:?

2. You can then edit your lesson by means of the editor commands E, D, L, U, R, T as explained on the following pages and exit by typing / in answer to the OPERATION:? prompt.

E COMMAND - TO ENTER LINES IN A LESSON

To begin entering lines in a new lesson type E in response to the OPERATION :? prompt. The computer will respond with 10 (indicating line number 10). Pressing return at the end of each line will cause the computer to respond with a new line number, incrementing by 10. For example:

```
OPERATION :?
10 Q. WHO DISCOVERED AMERICA?
20 A. COLUMBUS
30 R. CORRECT, IN JULY 1492.
```

To add lines to the end of an existing lesson type E in response to OPERATION :? The computer will respond with a line number ten greater than the last line number in the lesson.

To insert lines within an existing lesson type E and a number between two existing line numbers. For example, to insert a new line between lines 20 and 30 in the lesson above, type E25. The computer will respond 25.

To replace an existing line in a lesson, type E and the line number to be replaced. For example, to replace line 30 in the above example, type E30 in response to OPERATION :?. The computer will prompt 30.

To stop entering lines type a / in response to a line number prompt.

COMMENT

The E command takes the form E[n_1] [Sn_2] where n_1 is the starting line number and n_2 is the increment. If E is typed, the computer will automatically establish a default increment of 10, go to the last line of the lesson, increase the line number by the increment, and prompt you with this line number. EL will start you directly on the last line.

NOTE: THE MAXIMUM NUMBER OF LINES WHICH A SINGLE LESSON MAY CONTAIN IS 500. HOWEVER, IT IS STRONGLY RECOMMENDED THAT YOU RESTRICT YOUR LESSONS TO ABOUT 250 LINES. IF YOU HAVE A LESSON LONGER THAN THAT, BREAK IT INTO PARTS AND USE THE CHAIN COMMAND.

D COMMAND - TO DELETE LINES

To delete one line from a lesson, type D and that line number in response to the OPERATION :? prompt.

To delete several lines from a lesson, type D, the first line number of the section to be deleted, a hyphen and then the last line of the section to be deleted.

Examples:		
	To delete line 30, type	D30
	To delete lines 30 to 60, type	D30-60
	To delete the last line only, type	DL
	To delete the entire file, type	D-L

Caution is advised when using the D command. Be sure the line you delete is the line you wish to delete, for once gone it cannot be recovered. It is therefore advisable to list lines to be deleted before deleting them.

COMMENT

The D command takes the form D[n_1]-[n_2] where n_1 is the first line to be deleted and n_2 is the last line to be deleted.

L AND U COMMANDS - TO LIST THE LINES OF A LESSON

To list the first line of a lesson, type	L
To list all lines of a lesson, type	L-L
To list the first line through to line 50, type	L-50
To list lines 30 through to 100, type	L30-100
To list line 50 only, type	L50
To list only the last line of the lesson, type	LL

COMMENT

The L command takes the form L[n_1]-[n_2]where n_1 is the beginning line number and n_2 is the ending line number.

A complete listing is helpful if you wish to inspect your entire text after making several corrections on a practice lesson.

The U command works exactly like the L command except that it produces an unnumbered printout.

R COMMAND - TO RENUMBER LINES IN A LESSON

To renumber the lines in a lesson, type R in response to OPERATION :?. This will number the first line in a lesson 10, and each subsequent line 10 more than the previous line.

The renumber command is particularly useful when a section is to be inserted within an existing lesson. For example, suppose 15 lines of text were to be inserted between lines 10 and 20 of an existing lesson. This could be accomplished by inserting the first nine lines of new text as lines 11 to 19, then renumbering (line 11 would then become line 20, line 12 would become line 30, and line 19 would become new line 100). The next 6 lines of the new text could then be entered as lines 101 to 106.

COMMENT

The R command takes the form R[n_1]S[Sn_2] where n_1 is the beginning line number and n_2 is the increment. You may renumber your line numbers by typing R14S3 and end up with lines numbered 14, 17, 20, 23, etc.

If you wish an increment of 10 (the default increment), type R and the line number you wish to begin your renumbering with. A beginning line number of 10 is also a default. So typing R is equivalent to typing R10S10.

T COMMAND - TO TRANSFER LINES WITHIN A LESSON

To move or transfer lines from one place in your lesson to another type T and the line or lines to be transferred in response to OPERATION :?.

Examples:

```
T100       will move line 100
T200-650   will move all lines 200 to 650 inclusive
T800-L     will move all lines from line 800 to the last
              in the file
```

You will then be asked:

```
WHERE TO?    (line number)
```

and should then type in a line number specifying the destination for the transferred lines. The line or lines being moved will be inserted into the lesson file at a point after the destination line that you specify,, with the exception that a destination of Ø will transfer the lines to before the first line in the file. To transfer to the end of lesson, type L.

Finally, the reorganized file is renumbered starting with 10 and incrementing by 10 as if you had issued an R command.

SUMMARY OF EDITOR COMMANDS

E — to enter or add lines to your lesson

E prompts you with the next incremental line number after the last line in your lesson

En deletes (erases) line number n (if it exists) and prompts you with line number n.

D — to delete lines from your lesson

Dn will delete line n

Dn-m deletes all lines from line n to line m

DL deletes the last line in your lesson

D-L deletes all lines in your lesson (wow!)

L — to list lines in your lesson

L lists the first line in your lesson

Ln lists line n

Ln-m lists all lines from line n to line m

LL lists the last line in your lesson

L-L lists all lines in your lesson

U — to list lines of your lesson without line numbers (works just like L)

R — to renumber the lines in your lesson

R renumbers all lines in your lesson with a beginning line number of 10 and an increment of 10

RnSm renumbers line n and all subsequent line numbers with an increment of m

T — to transfer lines from one place in a lesson to another

Tn moves line n

Tn-m moves lines n through m

N.B. Once you type the T command you will be asked "Where to?"

PART III

RUNNING COPILOT

Running a lesson.

To Run a COPILOT Lesson (Not Menu Driven)

When you are writing a COPILOT lesson it is easier to check out your lesson by using the prompt command from the Editor "DO YOU WANT TO RUN YOUR LESSON?" (see page 22). Alternatively, you may wish to run your lesson directly, as shown below.

1. You type — RUN COPILOT

2. The computer will type —
   WELCOME TO APPLESOFT COPILOT
   WHICH LESSON WOULD YOU LIKE TO STUDY?

   Type in the name of the lesson you want.

3. The computer will type - and will give you the lesson after a 10-15 second delay — JUST A MINUTE FOR LESSON <NAME>

4. When the lesson is finished the computer will type — DO YOU WANT TO STUDY ANOTHER LESSON?

   Type — YES or NO

5. If you type YES the computer will return to step 2, above.

6. If you type NO the computer will type —
   THANK YOU FOR STUDYING THIS LESSON.
   GOODBYE FOR NOW.
   HOPE TO SEE YOU AGAIN.

7. To stop running a lesson, press the <RESET> key.

8. You may enter a lesson at any location (L.) by typing the name of the lesson, a semi-colon and the location number, for example:

   EXPLOR;37

   will cause the lesson EXPLOR to begin at location 37. This is a particularly valuable feature when you are editing a section of an existing lesson and want a test run on that section.

MENU DRIVEN COPILOT

Overview

The two programs HELLO1 and CREATE MENU allow you to create a turn-key, menu driven, lesson disk. With this disk the student simply boots the disk by turning on the power (Apple II Plus or Apple II with autostart ROM) or typing PR#6 or C600G (Apple II without autostart ROM). A menu (list of lessons) is displayed on the screen. The student selects the lesson by typing the number in front of the lesson name. The lesson is automatically loaded and run. When the lesson is over the menu is once again displayed. Note - Because neither COPED nor CREATE MENU are on the lesson disk, it is not easy for a mischievous student to alter either a lesson or the menu.

To Create a Turn-key System

Once you have your lessons in a form where you are ready to create the Turn-key disk follow the steps below.

1. Load the program HELLO1 from the COPILOT master disk into the Apple's memory.

2. Initialize a blank disk with this program by typing

   INIT HELLO

   Note, this program contains the turn-key version of COPILOT so you do not need a separate version of COPILOT on your lesson disk. Hereafter we shall refer to this disk as the "Turn-key disk".

3. Copy the program CREATE MENU from your COPILOT master disk onto the Turn-key disk using FID (which is on your DOS 3.3 System Master disk). Alternately, load the program into memory, replace the COPILOT master disk with the Turn-key disk, and type SAVE CREATE MENU.

4. Copy each of your lessons (i.e. COPILOT files) to the Turn-key disk using FID. Alternatively you can use COPED to call a lesson file into memory, replace the COPILOT Master Disk with the Turn-key disk, and save the lesson by typing a / to the prompt Operation? .

5. Run the program CREATE MENU. This program will lead you gently through the steps for creating a menu of the lessons. If you are creating a new menu, CREATE MENU will set up the file COPILOT.MENU.FILE and will store the description of the lesson (up to 26 characters) and the actual lessonfile name. CREATE MENU also allows you to add lessons to an existing COPILOT.MENU. FILE. Again, the instructions are contained in the program.

6. Once you are satisfied with your menu, delete CREATE MENU and LOCK all of the remaining programs and files. The catalog of your Turn-key disk should now have only the following programs:

```
*A 032 HELLO
*T 004 COPILOT.MENU.FILE
*T (COPILOT LESSONS)
*T (                )
*T (                )
 . (                )
 . (                )
 . (                )
 . (                )
```

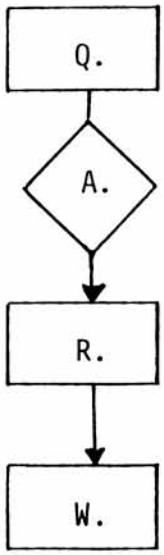
PART IV

LESSON EXAMPLES

LESSON EXAMPLE 1

QUESTION AND ANSWER - NO SECOND TRIES

Flowchart:



Text and question

Accept student responses and compare with expected 'right' answer.

Teacher reply if student answer is 'right'. Otherwise this step will be ignored.

Teacher reply if student answer is 'wrong' - otherwise this step will be ignored.

LESSON CONTINUES

Example:

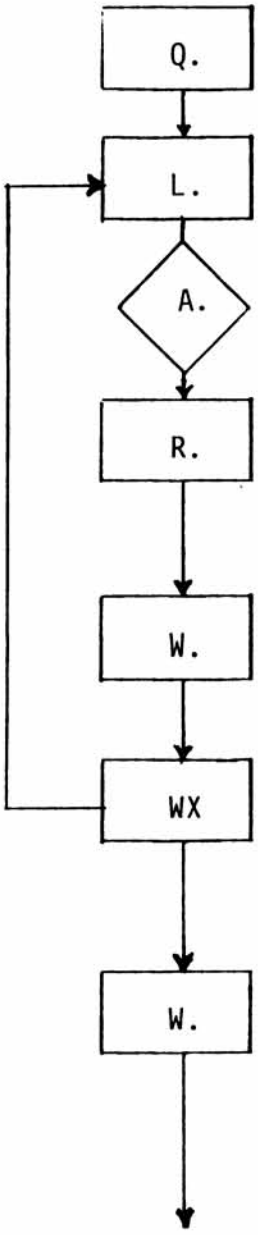
```
Q. THE MAJOR FACTOR IN THE MAKING OF
Q. RICH HUMUS SOILS IS
Q.      WATER              GRAZING
Q.      BACTERIA           CULTIVATION
Q. TYPE YOUR ANSWER
A. BACTERIA
R. VERY GOOD.
W. WATER, GRAZING, AND CULTIVATION ALL CONTRIBUTE TO THE MAKING
W. OF HUMUS, BUT THE ACTION OF BACTERIA ON ORGANIC MATTER
W. SUCH AS LEAVES AND FALLEN TREES IS THE MOST
W. IMPORTANT FACTOR
```

(lesson continues)

LESSON EXAMPLE 2

QUESTION AND ANSWER - STUDENT GIVEN A SECOND TRY

Flowchart:



Q. — Text and Question

L. — Label n

A. — Accept student response and compare with expected 'right answer'.

R. — Teacher reply if student answer is 'right' otherwise this step will be ignored.

W. — Teacher response if student response is 'wrong' otherwise this step will be ignored.

WX — Transfer to label n if student response 'wrong' WRONG

RIGHT

W. — Teacher response if student response 'wrong' on second try. If student response 'right' on either first or second try, this step will be ignored.

Example:

```
Q. THE POEM I WANDERED LONELY AS A CLOUD WAS
Q. WRITTEN BY WHICH OF THE FOLLOWING POETS
Q. T.S. ELIOT       WORDSWORTH
Q. HOUSEMAN         MASEFIELD
L.3
A. WORDSWORTH
R. ABSOLUTELY CORRECT!  WORDSWORTH WROTE IT IN 1804.
W. NO - BUT LET ME GIVE YOU A HINT.  HE
W. WAS THE EARLIEST OF THE FOUR POETS.
WX3
W. WELL, YOU CAN'T WIN THEM ALL.  WORDSWORTH, THE
W. CORRECT ANSWER, WROTE IN A MORE PROSAIC STYLE
W. THAN THE THREE LATER POETS.
W. HOUSEMAN LIVED FROM 1859 TO 1936, MASEFIELD
W. FROM 1874 TO 1967, AND ELIOT FROM 1888 TO 1965.
```

(lesson continues)

COMMENT

The placement of the L.3 just before the A. is critical. If it were placed before the Q.s listing the poets the W. message 'No - but ...' would be repeated if the student gave the wrong answer on the second try. If it is placed after the Q.s, and the student gives an incorrect response on the second try, only the series of W.s beginning with 'Well, you can't win ...' is printed. The way COPILOT functions is that if a WX, W., RX, or R. has once been executed, it will not execute again <u>until</u> a Q. has been executed.

LESSON EXAMPLE 3

Subroutines

```
10 Q.THIS IS A LESSON ABOUT ELECTRICITY.
20 Q.LET US SEE WHETHER YOU KNOW THE
30 Q.BASIC UNIT USED IN ELECTRICITY.
40 Q.
50 Q.WHAT IS THE UNIT OF MEASURE OF
60 Q.RESISTANCE?
70 A.@OHM@
80 R.CORRECT YOU ARE!
90 W.
100 WX1
110 Q.TRY ANOTHER.
120 Q.WHAT IS THE BASIC UNIT OF MEASURE
130 Q.OF CURRENT?
140 A.@AMPERE@
150 R.GOOD THINKING!
160 W.
170 WX1
180 Q.ONE MORE QUESTION.
190 Q.WHAT IS THE UNIT OF MEASURE OF
200 Q.ELECTROMOTIVE FORCE?
210 A.@VOLT@
220 R.YOU GOT IT!
230 W.
240 WX1
250 Q. THIS ENDS THIS SHORT LESSON.
260 X.99
270 ******** END OF LESSON *********
280 **
290 *****SUBROUTINE*******
300 L.1
310 Q.LET ME GIVE YOU SOME HELP.
320 Q.
330 Q.THE UNIT OF MEASURE OF ELECTROMOTIVE
340 Q.FORCE IS THE VOLT.
350 Q.THE UNIT OF MEASURE OF CURRENT IS
360 Q.THE AMPERE.
370 Q.THE UNIT OF MEASURE OF RESISTANCE
380 Q.IS THE OHM.
390 Q.PRESS <RETURN> TO GO ON.
400 Z.
410 HM
420 X.0
430 ******* END OF SUBROUTINE *********
440 L.99
450 Q.BYE FOR NOW LITTLE HUMAN.
```

COMMENT

This example shows the use of subroutines, that is a section of a lesson which can be used over and over again. In the short example above, the subroutine is between lines 290-430.

When student answers incorrectly any of the three questions, the subroutine is "called", that is there is a jumpt to the subroutine. This is accomplished by the WX1 commands at lines 100, 170, and 240. The W. at lines 90, 160, and 230 are to suppress the default printing of WRONG.

The subroutine prints the text following the Q.s in lines 310-390. When the student presses <RETURN> as directed in line 390, the screen is cleared with the HM command (see the advanced COPILOT commands later in this manual). The X.0 at line 420 jumps (returns) to the line following the last transfer.

Lines 270, 280, 290, and 430 are not processed. Only lines beginning with a valid two character command are processed; all other lines are ignored when the program is run. These lines are included as documentation for the author of the lesson.

Line 260 causes the lesson to skip over the subroutine when the lesson is run.

HINTS

```
10 Q.       EXPLORERS
20 Q.
30 Q.THIS IS A LESSON ABOUT EXPLORERS.
40 Q.WHO DISCOVERED AMERICA?
50 Q.IF YOU WANT A HINT TYPE //HINT
60 L.3
70 A.@COLUMBUS@
80 R.  FANTASTIC!
90 RX2
100 A,//HINT
110 R.
120 W.
130 RX1
140 W.NO.  TRY AGAIN
150 WX3
160 W.STILL WRONG.  THE ANSWER IS
170 W.       COLUMBUS
180 X.2
190 ********SUBROUTINE FOR HINT**********
200 **
210 L.1
220 Q.HE WAS AN ITALIAN WHO SAILED FROM
230 Q.SPAIN IN 1492.
240 Q.WHAT DO YOU THINK HIS NAME WAS?
250 X.3
260 ********END HINT SUBROUTINE********
270 L.2
280 Q.      LESSON CONTINUES
```

COMMENT

Line 70 causes the computer to accept a student response. If he types Columbus the 'rightness' condition is established at line 70, the comment in line 80 (FANTASTIC!) is printed for the student and the RX2 transfer is executed to line 270, and the lesson continues at line 280.

If the student types //HINT the condition of rightness is established at line 100. The RX1 is executed at line 130 to line 210.(Lines 110 and 120 suppress the default printing of the words 'RIGHT' and 'WRONG'.) The hint at lines 210-240 is typed and transfer X.3 is executed back to line 60 to permit the student to respond, now that he has a hint.

If the student responds something other than Columbus or //HINT at line 70 line 140 is executed and he is transferred back to line 60 (L.3) through the WX command in line 150.

LESSON EXAMPLE 5

Use of 'Wrong' Response for Prompting

```
 10 Q. THIS IS A QUESTION ON GEOGRAPHY
 20 Q. WHICH IS THE MOST POPULOUS STATE?
 30 L.1
 40 A. NEW YORK
 50 A. NY
 60 R. NO, IT IS NOT NEW YORK
 70 R. ITS POPULATION HAS BEEN EXCEEDED
 80 R. BY ANOTHER STATE
 90 R. TRY AGAIN.
100 RX1
110 A, CALIFORNIA
120 A, CA
130 R. THAT'S RIGHT!
140 R. CALIFORNIA IS NOW THE
150 R. MOST POPULOUS STATE.
160 W. NO; I'M AFRAID YOU'RE WRONG.
170 W. TRY AGAIN.
180 WX1
190 W. STILL WRONG
200 W. IT'S A STATE ON THE WEST COAST.
210 W. TRY ONCE MORE.
220 WX1
230 W. THE ANSWER IS CALIFORNIA.
240 W. PLEASE BRUSH UP ON YOUR GEOGRAPHY.
250 W. GO ON TO THE NEXT QUESTION.
260 W. NEXT QUESTION
```

COMMENT

The terminal will pause at line 40 and wait for the student to respond. The A. in line 50 will be treated as an A, since it is the next command after another A.

If the student responds 'New York' in line 40 the answer is evaluated as 'Right' even though the teacher knows it is wrong. Therefore the statements to correct the student's wrong impression (lines 60-90) are preceeded by R. commands. In line 100 there is a conditional jump RX1 back to line 30 L.1 for the student to try again.

If the student typed CALIFORNIA at line 40, lines 60 to 100 would be ignored. At line 110, the student's previous answer at line 40 would be matched with 'CALIFORNIA' and 'CA'. (Note - the A, command is used at line 110 since, if an A. were used COPILOT would have waited for the student to respond again. A, command therefore is the proper command.

If the student did answer CALIFORNIA or CA, lines 130 to 150 will be typed and lines 160 to 250 will be ignored because each is a W. or WX command.

COMMENT (Cont'd)

If the student responded something other than NEW YORK or CALIFORNIA, lines 160 and 170 will be typed back.

Line 180 (WX1) says jump back to location 1 for another try. If, on the next try, the student still gets the answer wrong the computer will skip to line 190 (the line after the last WX transfer) and 190 to 210 will be typed. This skipping mechanism is built into COPILOT. The jump in line 220 will then be executed.

On the third try if the answer is still wrong, the computer will skip to line 230 (the line after the last WX transfer) and lines 230 to 250 will be typed.

If the student eventually gives the right answer before his three tries are exhausted, lines 130 to 150 will be typed. In any case, after a correct answer, or three tries, the next question will be asked.

PART V

ADVANCED COPILOT COMMANDS

This section describes the features of COPILOT which use the graphics and sound producing capabilities of the Apple II.

NT and NS

(The musical NoTe and Note Sound commands)

The NoTe command causes the speaker inside of the Apple to produce a musical tone. The format of the command is NTn1,n2 where n1 is the pitch and n2 is the length of the tone. The values for n1 are in the range 0-37, where a value of 6 is about middle C. The scale is more or less a semi-tone scale which varies a bit from Apple to Apple and so you will need to do a bit of experimenting before you sit down at the Apple keyboard and write your first concerto. n2 determines the length of the note. Again the length varies a bit from Apple to Apple so do a bit of experimenting. The value of n2 can be any number from 1 to 255. If n1 = 0 then a rest of length n2 is generated.

The Note Sound command, NS, can be used to add sound effects to your lesson. The format of the command is NSn1,n2 where n1 determines the number of 'Sounds' and n2 determines the length of the 'Sounds'. Depending on the value of n2 the sound varies from a sharp click to a sort of a 'whoop' sound. The values of n1 and n2 can be any number from 1 to 255.

COMMENT

A series of notes or sounds may be made using a single NS or NT command. The format is NSn1,n2,n3,n4.... where n1 and n2 identify the first sound, n3 and n4 identify the second sound etc. A similar format is used for the NT command.

The following demonstrates the NS and NT commands.

```
10 SP50
20 Q.FIRST A MIDDLE C
30 NT6,100
40 Q.A LONGER MIDDLE C
50 NT6,200
60 Q.A SERIES OF NOTES
70 NT6,100,8,100,10,100,12,100,14,100
80 Q.NOW FOR SOME SOUNDS
90 Q.
100 Q.NS1,25 GIVES A CLICK
110 NS1,25
120 Q.NS1,100 GIVES A ZIP
130 NS1,100
140 Q.NS1,255 GIVES A WHOOP
150 NS1,255
160 Q.NS10,50 GIVES A BUZZ
170 NS10,50
180 Q.NS 10,200 ZAPS YOU
190 NS10,200
200 THAT'S ALL FOLKS
210 SP255
```

VT, HT and HM

(The Vertical Tab, Horizontal Tab and HoMe commands)

The screen of the Apple has 24 lines of 40 characters each. The Vertical Tab and Horizontal Tab commands (which operate in text mode, as opposed to the graphics mode described later in this section) position the line of text which follows a VT or HT command.

The formats for these two commands are: VTn and HTn where n is a number between 1 and 24 for VT and between 1 and 40 for HT.

The Horizontal Tab command, HT is like the TAB key on a typewriter. HT9 would start the next line of text 9 spaces indented.

The Vertical Tab command, VT, operates like the Horizontal Tab command except that it tabs the line of text down the screen.

The HoMe command, HM, clears the screen and causes any subsequent text to be printed beginning on the top line of the screen.

COMMENT

The use of the VT may be demonstrated by the following segment of a lesson:

```
10 HM
20 VT12
30 Q.THIS IS A LESSON ABOUT POETS.
40 Q.LET US BEGIN WITH OGDEN NASH.
50 Q.PRESS<RETURN>TO END
60 Z.
```

In this example the statement 'THIS IS A LESSON ABOUT POETS' would be printed halfway down the screen, at line 12. The line 'LET US BEGIN WITH OGDEN NASH.' would be printed immediately following the line 'THIS IS A LESSON....' since this would be the normal printing procedure.

It is important to note that VT overrides the normal sequence of printing. For example, the lesson shown below would first cause 'STATEMENT 1' to be printed on the 12th line of the screen, 'STATEMENT 2' to be printed on the 5th line of the screen and then 'STATEMENT 3' on line 12.
(N.B. Line 5 slows down the printing speed so you can see what is happening. Line 95 restores the speed to normal. The speed command is explained in the next section.)

```
5 SP50
10 HM
20 VT12
30 Q.STATEMENT 1
40 VT5
50 Q.STATEMENT 2
60 VT12
70 Q.OVERWRITE STATEMENT 1 WITH 3
80 Q.PRESS<RETURN> TO END
90 Z.
95 SP255
```

IN, NO, and FL

(INverse, NOrmal, and FLash)

The INverse command causes the printing on the screen to be black letters on a white background instead of the normal white letters on a black background. The format of the command is simply IN.

The NOrmal command returns the printing on the screen to be white letters on black background. The format of the command is simply NO.

The FLash command causes the printing on the screen to 'flash', that is alternatively be white characters on a black background and then black characters on a white background. The format of the command is simply FL. The NOrmal command stops the flashing.

COMMENT

The following demonstrates the INverse, NOrmal and FLash commands.

```
10 Q.FIRST TURN ON THE FLASH
20 FL
30 Q.PRESS<RETURN>TO CONTINUE
40 Z.
50 Q.BACK TO NORMAL
60 NO
70 Q.NOW FOR THE INVERSE
80 IN
90 Q.PRESS<RETURN> TO CONTINUE
100 Z.
110 Q.BACK TO NORMAL
120 NO
```

SP, CO, and HC

(The SPeed and COlour commands)

SP sets the SPeed of printing on the screen of the Apple. The format of the command is SPn where n is from 1 to 255. At SP255 the printing is the normal screen printing speed of the Apple - much too fast to read. At SP100 the printing is at about 150 words per minute. At SP50 the printing is at about 100 words per minute. Note, after a SP command the speed of printing is set at that level until it is reset by another SP command.

The CO sets the COlour of low resolution graphics on the screen. The format for the COlour command is COn where n is a number between 0 and 15 which defines the colour. The colour names and their associated numbers are:

0	black	8	brown
1	magenta	9	orange
2	dark blue	10	grey
3	purple	11	pink
4	dark green	12	green
5	grey	13	yellow
6	medium blue	14	aqua
7	light blue	15	white

Note: Depending on what colour TV set you have and how it is adjusted, the colours may be different from those shown above.

The High resolution Colour command, HC sets the colour in high resolution graphics. The format is similar to the CO command except that fewer colours are available and the colours vary depending on whether the coordinates are odd or even (consult an Applesoft manual for an explanation) and the TV set used.

The COPILOT commands on this, and the preceeding two pages are equivalent to Applesoft commands as follows:

VT	*VTAB*
HT	*HTAB*
HM	*HOME*
IN	*INVERSE*
NO	*NORMAL*
FL	*FLASH*
SP	*SPEED*
CO	*COLOR*
HC	*HCOLOR*

GR, PL, HL, VL, TX

(GRaphics - The Low Resolution Kind)

The GRaphics command, GR, sets low resolution graphics mode. This provides the facility for making simple line drawings on a 40 x 40 grid at the top of the screen with four lines of text at the bottom. Each of the 1600 points on the 40 x 40 grid may be turned on (i.e. made bright) using the PL command once graphics mode is set by the GR command.

The PLot command, PLn1,n2 causes the point on the screen with the X coordinate of n1 and the y coordinate of n2 to be turned on. The upper left hand corner of the screen is 0,0 (i.e. n1 = 0 and n2 = 0). The upper right hand corner is 39,0 (if you start counting at 0 instead of 1, 40 points is equal to 0 through 39). The lower right hand corner is 39,39. The middle is either 19,19 or 20,20 - we are not sure which because we learned to count starting with 1 instead of 0.

The Horizontal Line command, HLn1,n2,n3 draws a horizontal line on the screen from n1 to n2 at n3. For example, HL1,9,19 would draw a horizontal line about ¼ of the way across the screen beginning near the left side about ½ way down the screen. HL0,39,0 would draw a horizontal line across the top of the screen. HL0,39,39 would draw a horizontal line near the bottom of the screen just above the first of the four lines of text.

The Vertical Line command, VL, is similar to HL except that the line is horizontal. The format is VLn1,n2,n3. The command VL10,20,39 will draw a vertical line from vertical grid point 10 to vertical grid point 20 at the extreme right hand side of the screen.

The TeXt command, TX, clears the screen and turns off graphics mode.

COMMENT

The GR command sets the colour to white (CO15). If a different colour is desired, use the CO command after the GR command.

A series of points may be turned on using a single PL command. The format is PLn1,n2,n3,n4 ..., where n1 and n2 identify the first point to be turned on, n3 and n4 identify the second point to be turned on etc.

A series of horizontal lines also may be turned on with a single HL command. The format is HLn1,n2,n3,n4,n5,n6 ...., where n4,n5 and n6 identify the second line in the same way that n1,n2 and n3 identify the first line, etc.

In a similar manner a series of vertical lines may be drawn with a single VL command.

To cancel the effect of the GR command and reinstitute the normal screen format of 24 lines of 40 characters each, use the TX command.

The following series of commands illustrate the use of the graphics commands and will draw a small yellow (we hope) rectangle in the centre of the screen.

```
10 GR
20 SP50
30 CO13
40 HL16,24,17,16,24,22
50 VL18,21,16,18,21,24
60 Q.PRESS <RETURN> TO CONTINUE
70 Z.
80 TX
90 SP255
```

Line 10 turns on the graphics mode and clears the screen.

Line 20 slows the speed down so you can see what is happening.

Line 30 sets the colour to yellow.

Line 40 draws two horizontal lines - both from gridpoint position 16 to 24, one at horizontal position 17, and the other at horizontal position 22.

Line 50 draws two vertical lines - both from gridpoint position 18 to 21, one at vertical position 16 and the other at vertical position 24.

Line 60 prints a line of text at the bottom of the screen.

Line 79 keeps the image on the screen until you press a key.

Line 80 returns the screen to normal text mode.

Line 90 returns the speed to normal.

COMMENT

If you decide to PLot a series of points using a single PL command, use spacing (which does not affect the command) to help you keep track of what you are doing. For example, the command

```
PL   5,10,   10,15,   15,20,   20,25,   25,30
```

conveys a much clearer image of what will appear on the screen than the command

```
PL5,10,10,15,15,20,20,25,25,30
```

In a similar manner, you can use spacing with the HL and VL commands (which we deliberately did not do above to make the point).

HG and HP

(High Resolution Graphics)

The High resolution Graphics command, HG, sets the colour to white and sets high resolution graphics mode: a 280 x 160 point grid and 4 lines of text at the bottom of the screen. Each of the 44,800 points may be individually turned on using the HP command.

The High resolution Plot command, HP, either causes a single point to be turned on or draws line(s). The command HPn1,n2 will cause point at gridpoint n1,n2 to be turned on. The command

```
HPn1,n2,    n3,n4
```

will cause a line to be drawn from n1,n2 to n3,n4. The command

```
HPn1,n2,    n3,n4,     n5,n6
```

will cause lines to be drawn from n1,n2 to n3,n4 and thence to n5,n6.

As an example of the HP command, the following draws a triangle

```
10 HP75,0,    0,150,    150,150,    75,0
```

COMMENT

If you wish to set the colour of your drawing in high resolution graphics, set the colour using the HC command before the HP command. As indicated earlier the HC command is equivalent to the Applesoft HCOLOR command. The Applesoft manual indicates the colour names and their associated values are:

0 black 1
1 green (depends on TV)
2 blue (depends on TV)
3 white 1
4 black 2
5 depends on TV
6 depends on TV
7 white 2

You should consult your Applesoft manual for further information.

The SHAPE MAKER

While the HP command is convenient for drawing straight lines or turning on single points, it is cumbersome for irregular shapes. An alternative to the HP command is to use the SHAPE MAKER to produce a shape table.

What is a shape table, you might ask. A shape table is simply a binary file (as opposed to an Applesoft program) which can be stored on disk. This file contains a series of plotting vectors which will produce a shape on the screen when the LS (Load Shape) and HD or HX command are included in a COPILOT lesson. Each shape produced is in high resolution graphics and has a maximum size of 75 x 75 points. Since the high resolution graphics screen is 280 x 160 points (plus 4 lines of text at the bottom) this means the largest single shape which can be produced is a bit larger than ¼ of the height of the screen and a bit smaller than ½ of the width. But wait! You can use the High resolution Scaling command to increase the size of your shape, but more about that later. You can also display more than one shape on the screen at a time and so it is possible to put several shapes together to fill the screen with continuous lines. Note that each shape table can contain a number of shapes. The exact number will depend on the complexity of each drawing.

To use the SHAPE MAKER type

```
RUN SHAPE MAKER
```

The following menu will appear on the screen:

```
       ::: COPILOT SHAPE MAKER :::
----------------------------------------

      1/   INITIALIZE NEW SHAPE TABLES
      2/   DRAW SHAPES
      3/   CATALOG DISK
      4/   VIEW OLD SHAPES
      5/   HELP!
      6/   END PROGRAM
        SELECT=>
       ::: COPILOT SHAPE MAKER :::
----------------------------------------
```

If you typed 5 you would see the following message:

```
THIS PROGRAM WILL ENABLE THE CONSTRUCTION OF SHAPE-
TABLES FOR USE IN 'COPILOT' PROGRAMS.  THE SHAPE TABLES
ARE SAVED ON DISK UNDER A NAME NOMINATED BY THE USER.
OVER TWO HUNDRED SHAPES MAY BE SAVED IN ONE SHAPE TABLE,
BUT THE TOTAL TABLE SIZE IS LIMITED TO A FEW THOUSAND
BYTES.  SHAPE TABLES MAY BE SAVED TO DISK EVEN THOUGH
INCOMPLETE.  FOR EXAMPLE, IF YOU DECIDE TO CREATE A
TABLE WITH TEN SHAPES, IT IS POSSIBLE TO ONLY DRAW TWO
AND THEN TO SAVE IT.
IF YOU RELOAD THAT SHAPE TABLE LATER, THE FIRST SHAPES
YOU DREW WILL STILL BE THERE.  YOU CAN THEN ADD THE
REMAINING SHAPES.
```

```
TO START A NEW SHAPE TABLE, CHOOSE OPTION '1'.  THIS WILL ALLOW
YOU TO NAME THE NUMBER OF SHAPES THE TABLE MAY HOLD.  THIS NUMBER
MAY BE LARGER THAN YOU EXPECT TO NEED:  IT IS WISE TO HAVE ROOM
FOR A 'FORGOTTEN' SHAPE.
FOR EXAMPLE, YOU MAY NEED 10 SHAPES IN A PROGRAM.  IT IS QUITE
REASONABLE TO DEFINE THE MAXIMUM NUMBER OF SHAPES AS 15 OR 20.
AFTER USING OPTION '1', USE OPTION '2' TO ACTUALLY DRAW THE
SHAPES.

IF YOU HAVE A SHAPE TABLE SEMI-COMPLETE THAT HAS BEEN SAVED ONTO
DISK, USE OPTION '2' TO CONTINUE WORKING ON FROM WHERE YOU LEFT
OFF.
NOTE THAT YOU DO NOT HAVE TO FILL THE SHAPE TABLE TO USE IT IN A
PROGRAM.
HIT RETURN >
```

After you have read the instructions on the screen, you will then need to initialize a shape table (as it says in the instructions) so in response to the prompt

```
SELECT=>
```

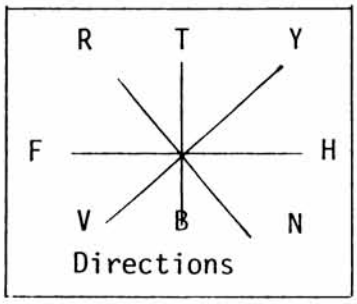
type 1.

A set of instructions will appear on the screen and the cursor will flash after the word NAME: Give it a name: it is a good idea to indicate in the name that it is a shape table (e.g. SHAPE TABLE 1 or ELECT.SHAPE). Then indicate how many shapes you expect to include in your table (plus a few extra). Press <RETURN> and the disk light will go on for a few seconds and the menu will reappear.

You are now ready to draw shapes, so type 2.

You will see a new set of instructions. The first instruction allows you to add shapes to any initialized shape table. If you wish to add to the table which you just created, simply press <RETURN>

Now we come to the exciting part! A square will appear on the left side of the screen and on the right side a box that looks like this



At the bottom of the screen is the message: "Move the dot to a starting position and press <RETURN> when you are ready to begin drawing your shape". The box gives you the clue as to how to move the dot in the left hand square. Press T and the dot moves up. Press B and it moves down.

Once you have the dot in the starting position, press <RETURN>. Then press P to turn the plot mode on. You can now draw shapes. When you press any of the keys shown in the box a dot will appear, and another and another. You can also hold the repeat key down to produce a line of dots. You can turn the Plot off by pressing P and then move the dot around the screen, turn it on at a new point (by pressing P) and plot a bit more, off again, move the dot on again etc. If you make a mistake, press E and it will erase the last dot. Press it again and it will erase the penultimate dot. Keep pressing E and you can erase everything you have just drawn.

Just below the box you will see the statement: "BYTES FREE n". As you draw a shape the value of n will get smaller. You can draw to your heart's content up until the time that the value of n is 0. Then you must quit.

When you have finished drawing your shape press Q, for quit. Two shapes will appear in the square, with the word DRAW above one and XDRAW above the other. These two shapes are what you would see in a COPILOT lesson if you used the commands HD and HX, respectively. The shapes are smaller than the original which you drew because the original was double size so you could draw in finer detail.

The following message has also appeared at the bottom of the screen: "Is this shape correct?" If you are not satisfied, type N, and you can start over. If you are satisfied, type Y and the computer will ask you if you are finished drawing shapes. If so, again type Y. It will then ask you if you want to save the shape table on disk. Type Y again, the disk drive will whirl and the menu will appear on the screen. While it may seem a bit complex here, you will find the whole process is very friendly and civilized.

LS, HD, HS, HR and HX

LS shapetablename Loads a Shape table from disk into memory for subsequent use within a COPILOT lesson. For example, if the name of the shape table is MAPS, the command would be LSMAPS. The LS command may be inserted anywhere in the COPILOT program. Since it does take a few seconds to load a shape table, if only one shape table is used in the lesson it is preferable to place the LS command at the beginning of the lesson so as not to interrupt the flow of the lesson.

HDn1,n2,n3 High resolution Draws shape n, from a previously loaded shape table with the point at which you started drawing the shape at n1,n2. For example, if shape number 3 is a square the command HD3,140,10 was included in a COPILOT lesson, and you started drawing the square from the upper left hand corner, a square would be drawn with its left side in the centre of the screen (the screen is 280 points wide) near the top (the graphics portion screen is 160 points high). It is possible to draw a series of shapes with a single HD command. For example, the command

```
HD1,10,10,     3,50,50,     2,100,100
```

would draw shape 1 starting at 10,10; shape 3 starting at 50,50; and shape 2 starting at 100,100. Note the spacing of the command; it is not necessary to insert the extra spaces to group the three draws, but it does make clearer the intent of the command.

HSn, High resolution Scales a shape, that is, makes it larger or smaller. HS2 would make the shape twice as large, HS3 three times as large, etc. The value of n must be between 0 and 255. Note: HS0 is the maximum size - not a single point. (Why? Because that is the way that Applesoft works.)

HRn, High resolution graphics Rotation, causes the next shape to be drawn to be rotated. HR16 will cause the shape to be rotated 90° clockwise. HR32 will cause the shape to be rotated 180° clockwise, etc.

HXn1,n2,n3 erases a previously drawn shape (n1) located at points n2,n3 if the colour is white. (HX actually draws the shape in the complementary colour. Since the complement of white is black, HX 'erases' the figure. If the colour is other than white, strange things may happen. See the Applesoft Manual for a fuller explanation of XDRAW, which is the equivalent of the COPILOT HX command.)

HX can also be used to move a shape across the screen. The command

```
HXn1,n2,n3,     n1,n4,n5      .....
```

will cause shape n1 located at n2,n3 to be erased and re-drawn at n4,n5. Each successive group of three numbers will either erase a figure or draw a figure.

COMMENT

The following is a sample program to illustrate some of the above features. A shape table, MAPS was previously created using the COPILOT SHAPE EDITOR. The shape table contains two shapes.

```
 10 LSMAPS
 20 HG
 30 Q.THE FIRST MAP AT 80,100
 40 HD1,80,100
 50 Q.PRESS <RETURN> TO DELETE THE MAP
 60 Z.
 70  HX1,80,100
 80 Q.ZAP! (WELL, ALMOST) THE SHAPE IS GONE.
 90 Q.PRESS <RETURN> TO CONTINUE
100 Z.
110 Q.NOW THE SECOND MAP AT 140,80
120 HD2,140,80
130 Q.PRESS <RETURN> TO CONTINUE
140 Z.
150 HX2,140,80
160 HR16
170 HD2,140,80
180 Q.THE SHAPE ERASED AND REDRAWN
190 Q.WITH A 90 DEGREE ROTATION
200 Q.PRESS <RETURN> TO CONTINUE
210 Z.
```

For a fuller explanation of the use of the graphics commands, see the Applesoft manual. The COPILOT commands are equivalent to the Applesoft commands as follows:

HG	*HGR:COLOR=3*
HC	*HCOLOR*
HP	*HPLOT*
LS	*BLOAD*
HD	*DRAW*
HX	*XDRAW*
HS	*SCALE*
HR	*ROT*
TX	*TEXT:HOME*
LP	*BLOAD*

GR	*GR:COLOR=15*
PL	*PLOT*
HL	*HLIN*
VL	*VLIN*
TX	*TEXT*

LP

The Load Picture command, LP, allows you to load and display a high resolution graphics picture which has been previously saved on disk. The format of the command is

```
LPname
```

where 'name' is the name of the picture. For example, suppose the picture was an outline map of Australia and the name of the picture was AUSTMAP.PIC. The following command would display the map on the screen

```
LP AUSTMAP.PIC
```

COMMENT

You can display a picture on the screen and then superimpose shapes by means of the LS command. For example, in the above example after displaying the outline map you could superimpose symbols showing major features like cities, mountains, etc. Note that when you display a picture it erases everything else on the screen so you cannot superimpose a picture over shapes, just the other way around.

There are a number of sources of pictures which can be used in your lessons. Your DOS3.3 system master contains some (at least the disk we received with our system did), and the Contributed Library (available from your authorized Apple dealer) contains many more. You can also draw your own using, for example, the Graphics Tablet. Fortunately it is not necessary to have a Graphics Tablet on your system to run a lesson with a picture drawn on the Graphics Tablet. All you need is a friend who will let you draw and store the picture on disk.

Alternatively you can draw your own picture on the first Hi-Res screen (HGR) using the Applesoft commands HCOLOR and HPLOT either in immediate execution mode or in a BASIC program. The picture may then be saved on disk using <u>either</u> of the following DOS commands (they are equivalent):

```
BSAVE <NAME>, A$2000,L$2000
```

or

```
BSAVE <NAME>, A8192,L8192
```

This picture may then be used in the same way as a picture from the Contributed Library or one produced using the Graphics Tablet.

It is a good idea to identify the fact that a particular binary file is a picture. We like the convention used by Mountain Hardware to add .PIC to the end of the name of the file.

(The POke Command)

The POke command, POn1,n2 allows the experienced Applesoft programmer to execute a POke command. For example, the COPILOT command

```
PO32,10,   33,10
```

will cause the left margin of the screen to be set at 10 and the width to be set at 10 characters. For a fuller explanation of POkes, consult the Applesoft Manual.

SUMMARY - ADVANCED COPILOT COMMANDS

Sounds:

NTn1,n2	Produces a musical tone of pitch n1(1-37) with length n2(1-255). NT0,n2 produces a rest of n2 length.
NSn1,n2	Produces a n1 sound (1-255) of n2 length (1-255)

Text Formatting:

VTn	Tabulates the next line of text to be printed n(1-23) lines down the screen.
HTn	Tabulates the next line of text to be printed n(1-39) across the screen.
HM	Clears the screen and causes the next line of text to be printed at the top of the screen.
IN	Causes the next line(s) of text to be printed in inverse mode (i.e. black letters on white background).
FL	Causes the next line(s) of text to flash.
NO	Returns the printing of text to normal.

Speed:

SPn	Sets the speed of printing. The maximum speed is n = 255. The minimum is n = 1.

Low Resolution Graphics:

GR	Sets the colour to white and sets low resolution graphics mode: a 40 x 40 grid at the top of the screen and four lines of text at the bottom.
COn	Sets the colour of the next shape to be printed. The value of n is from 0 to 15.
PLn1,n2	Causes the point on the screen with coordinates n1,n2 to be turned on.
HLn1,n2,n3	Causes a horizontal line to be drawn from n1 to n2 at vertical position n3.
VLn1,n2,n3	Causes a vertical line to be drawn from n1 to n2 at horizontal position n3.
TX	Clears the screen and turns off graphics mode.

High Resolution Graphics:

HG	Sets the colour to white and sets high resolution graphics mode: a 280 x 160 grid at the top of the screen and 4 lines of text at the bottom.
HCn	Sets the colour of the next shape to be printed. The value of n is from 0 to 7.
HPn1,n2	Draws a line from n1 to n2.
LSname	Loads shape table <name> into memory for subsequent use in a COPILOT lesson.
HDn1,n2,n3	Draws shape n1 from a previously loaded shape table starting at n2,n3.
HSn	Scales the next shape to be drawn. HS2 makes the figure twice as large, HS3 three times as large, etc.
HRn	Rotates the next shape to be drawn. If n = 16 the rotation is 90 degrees; if n = 32 the rotation is 180 degrees etc.
HXn1,n2,n3	If the colour is white erases shape n1 from the screen. If the colour is white, HXn1,n2,n3, n1,n4,n5 causes shape n1 at position n2,n3 to be erased and redrawn at n4,n5.
LPname	Loads picture <name> from disk and displays it on the screen.
TX	Turns off graphics mode and clears the screen.

Pokes:

POn1,n2	Pokes the value n2 into memory location n1.

APPENDIX

COPILOT Utilities

Speeding up your Lesson

Arithmetic Features

COPILOT UTILITIES

(N.B. Some of these programs were on the V2.1 disk. All of the programs contain easy to follow instructions. Simply type RUN and the name of the program.)

COPLIST	This Applesoft program will list a COPILOT lesson file on to the screen.
COPRINT	This Applesoft program will print a COPILOT lesson file on the line printer.
FILE CONVERTER	This Applesoft program attempts to convert COPILOT lessons which were written for an 80 column output to 40 columns. Each Q., R., and W. line which is longer than 40 characters is broken up into two shorter lines. Note - we say "attempts to convert" because the results are not always the same as if the lesson had been written in a 40 column format originally.
FILE APPENDER	This program allows you to append one COPILOT file to another. Note - the line numbers of the second file must start after the highest line number in the first file. The normal procedure is to use COPED to renumber the first file starting with 1 with an increment of 1 (R1S1). Then renumber the second file starting with 500 and an increment of 1 (R500S1). Then RUN FILE APPENDER. When the second file has been appended, use COPED to renumber the joined files starting with 10 and an increment of 10 (R).
COPILOT TRACE	This program can be used as a debugging aid to find logical errors in your COPILOT lessons. The program is similar to COPILOT except that each COPILOT line is shown as it is executed.
CREATE MENU	This program can be used to create a menu, or list of lessons, for menu driven COPILOT disk (see preceding section).

SPEEDING UP YOUR LESSON

Compared to some other author languages, COPILOT runs relatively fast. There are, however, some lesson writing techniques for making it run even faster.

1. When COPILOT was originally designed we made the decision that we wanted minimum delay within the lesson between the time the student pressed <RETURN> and the computer responded. The trade off is that when the lesson is loaded in addition to the time taken to load the lesson into memory there is a delay while COPILOT does some housekeeping, e.g. the lesson lines are sorted into an array and the label positions are sorted into another array. You can reduce this delay by using a single Q. to write two or three lines of text. For example, the following line

```
350 Q.THE UNIT OF MEASURE OF CURRENT
      FORCE IS THE AMPERE
      THE UNIT OF MEASURE OF RESISTANCE
```

   will create exactly the same text when run as the three lines

```
350 Q.THE UNIT OF MEASURE OF CURRENT
360 Q.FORCE IS THE AMPERE
370 Q.THE UNIT OF MEASURE OF RESISTANCE
```

   The reason is that elements of the array in which the COPILOT lesson lines are put can store up to 120 characters of text. If you do use this technique, just be sure to position the first letter of subsequent lines under the first letter of the first line.

2. Use subroutines sparingly. When you use a subroutine COPILOT scans the label array from the beginning to locate the position in the lesson where that label occurs. This takes time.

3. Keep your lessons relatively short - say less than 300 lines (the maximum size is 500 lines). If your lesson is longer than 300, break it into two parts and use the chain command.

ARITHMETIC FEATURES

COPILOT was originally written to enable teachers to write tutorial type lessons. In its simplest form we mean a lesson where a concept is presented through a section of text followed by a question to test the student's understanding of the concept. Depending on the student's response to the question he or she is either given a reinforcement, another try, or branched to a routine where the concept is explained further. One of the important features of a tutorial lesson is that the sequence of the sections of text and questions is very carefully structured.

In contrast to the tutorial type lesson, in a drill and practice lesson the concepts are taught prior to the lesson being run. Typically in a drill and practice lesson the student is presented a series of randomly generated problems. While some drill and practice programs contain algorithms which vary the difficulty of the problems depending on a student's responses, the problems are still randomly generated.

What has all of this to do with the COPILOT arithmetic features, you might ask. If you (the teacher) wish to write a drill and practice type lesson, do not use COPILOT. Write it in another language. For example BASIC, with its FOR .... NEXT, RND, and READ .... DATA statements, is a much better language for writing drill and practice lessons.

One further point, COPILOT was originally developed to enable teachers to write tutorial type lessons in the humanities. As such, it was developed as a string handling language rather than one which had arithmetic capabilities.

Nevertheless, even though you are not writing a drill and practice type COPILOT lesson, you may sometime wish to use some arithmetic operations in your lesson. We have, therefore, developed a series of arithmetic commands. Here we hasten to state that this is a first version of the arithmetic commands and that at some later date we may modify these. We would hope that these modifications would simply be enhancements or additions and that COPILOT lessons written under Version 2.2 would run on subsequent versions. However at this point in time we can give no guarantee that this will happen.

The new arithmetic features provide 10 internal number registers, identified by the digits 0 to 9. N.B. Register 0 is special, and is called the accumulator. It is the only register on which arithmetic operations can be performed. All of the other registers are simply places where numbers can be stored for future use.

The registers can be displayed with the #D command. For example, if all of the registers were empty the #D command would cause the following display:

```
REG 0 (AC) 0
REG 1      0
REG 2      0
REG 3      0
REG 4      0
REG 5      0
REG 6      0
REG 7      0
REG 8      0
REG 9      0
```

All of the other arithmetic commands have what are called "arguments" (the term is used in the mathematic sense as meaning "an independent variable").

Two kinds of arguments can exist, <NUMBER> and <REG> as shown below:

<NUMBER>	Any valid Applesoft constant, e.g. 25 or 3.5 or - 3 or even 2.21 E5.
<REG>	A register number (0 - 9). Note - the number must by typed in the first space following the command since a blank is taken as 0.

As an example of the above #+<REG> command adds the number in <REG> to the accumulator. Thus #+<5> would add the number in REGister 5 to the number in the accumulator. For example, if the number in REGister 5 were 25, and if the accumulator already had the number 12 in it, the new value of the accumulator would be 37 (because the accumulator accumulates).

Arithmetic Commands

#C<NUMBER>	Adds the constant NUMBER to the number in the accumulator. The original number in the accumulator is lost.
#R<NUMBER>	Replaces the number in the accumulator with a random number between 1 and NUMBER. The original number of the accumulator is lost.
#Z<REG>	Puts 0 into register REG. The original content of REG is lost.
#L<REG>	Loads (adds) the number in REG to the accumulator. The number in REG is unchanged.
#S<REG>	Saves the number in the accumulator in REG. The original number in REG is lost. The number in the accumulator remains unchanged.
#+<REG>	Adds the number in REG to the number in the accumulator. The number in REG remains unchanged.
#-<REG>	Subtracts the number in REG from the number in the accumulator. The number in REG remains unchanged.
#*<REG>	Multiplies the number in the accumulator by the number in REG and stores the result in the accumulator. The number in REG remains unchanged.

#/<REG>	Divides the number in the accumulator by the number in REG and stores the result in the accumulator. The number in the REG remains unchanged.
#^<REG>	Raises the number in the accumulator to the power of the number in REG. The number in REG remains unchanged. For example, if the number in the accumulator were 2 and the number in REG was 3, this command would change the value of the accumulator to 8.
#Q<REG>	Divides the number in the accumulator by the number in REG and replaces the original number in the accumulator by the integer part (quotient) of the result. The number in REG remains unchanged. For example, if the number in the accumulator is 11 and the number in REG is 2, this command would cause the value of the accumulator to be changed to 5.
#I<REG>	Adds 1 to the number in REG.

Input - Output Commands

#P<REG>	Prints the number in REG followed by a carriage return. The number in REG is unchanged.
#,<REG>	Prints the number in REG but no carriage return. The number in REG is unchanged.
#G<REG>	Gets a numeric input from the keyboard and replaces the number in REG with it.
#D<REG>	Displays the contents of all REGs; very useful in debugging.

Test Condition Commands

(These commands are used to examine the number in a register and to establish a condition of "rightness" or "wrongness" depending on the value of the number. After the condition of "rightness" or "wrongness" is established commands such as R., W., RXn, and WXn can be used to control the program flow in the normal way.)

?P<REG>	If the number in REG is positive, establishes a condition of "rightness"; otherwise "wrong".
?Z<REG>	If the number in REG is zero, establishes a condition of "rightness"; otherwise "wrong".

?N<REG>	If the number in REG is negative, establishes a condition of "rightness"; otherwise "wrong".